

Memory Layout Optimizations of Structure and Object

Golovanevsky Olga
golovanevsky.olga@gmail.com

Agenda

1. History
2. Transformations
3. Integration into LTO
4. Escape analysis
5. Current status and Development plan

pre-History - fwhole-program

1. M.Hagog thesis, supervised by D. Bernstein, Technion 2000
2. M.Hagog, C.Tice, 2005 gcc 1st version on branch
paper: "Cache Aware Data Layout Reorganization Optimization in GCC"
3. K.Zadeck, D.Berlin type-escape analysis, on mainline since 2006
4. 2007 struct peeling on struct-reorg-branch
paper: "Struct-reorg current status and future perspectives"
5. 2008 struct peeling on mainline, in gcc 4.3
6. struct-reorg is not supported on mainline from late 2008
7. disabled from mainline from gcc 4.7.0

History

1. LTO/WHOPR in from gcc 4.5.0
2. LTO - defines symbols resolution by asking linker - reduce number of escaping symbols
3. Sponsored by Cavium, A.Pinski, 2014-2015
4. New version of struct-reorg - LTO based, gcc 4.9.0
5. Spec2006: 462.libquantum, +30% on x86 (*Xeon*), up to +90% on arm (*ThunderX*)

Transformations - peeling

```
struct my_str {int a; double b;}
```

```
struct my_str *pStr;
```

```
n=10;
```

```
pStr = malloc(n*sizeof(my_str));
```

```
for (i=0; i < n; i++)  
    pStr[i].a = rand();
```

```
...  
for (i=0; i < n; i++)  
    pStr[i].b = rand();
```

```
struct my_str_1 {int a;}  
struct my_str_2 {double b;}
```

```
struct my_str_1 *pStr_1;  
struct my_str_2 *pStr_2;
```

```
n=10;
```

```
pStr_1 = malloc(n*sizeof(my_str_1));  
pStr_2 = malloc(n*sizeof(my_str_2));
```

```
for (i=0; i < n; i++)  
    pStr_1[i].a = rand();
```

```
...  
for (i=0; i < n; i++)  
    pStr_2[i].b = rand();
```

Transformations - peeling, struct frame removal

```
struct my_str_1 {int a;}  
struct my_str_2 {double b;}  
struct my_str_1 *pStr;  
struct my_str_2 *pStr;  
n=10;
```

```
pStr_1 = malloc(n*sizeof(my_str_1));  
pStr_2 = malloc(n*sizeof(my_str_2));  
for (i=0; i < n; i++)  
    pStr_1[i].a = rand();  
...  
for (i=0; i < n; i++)  
    pStr_2[i].b = rand();
```

```
int *pStr_a;  
double *pStr_b;  
n=10;
```

```
pStr_a = malloc(n*sizeof(int));  
pStr_b = malloc(n*sizeof(double));  
for (i=0; i < n; i++)  
    pStr_a[i] = rand();  
...  
for (i=0; i < n; i++)  
    pStr_b[i] = rand();
```

Transformations - struct inlining, 462.libquantum

```
struct quantum_reg_node_struct
```

```
    COMPLEX_FLOAT amplitude; /* alpha_j */
```

```
    MAX_UNSIGNED state; /* j */
```

```
};
```

peeling with frame removal



```
typedef struct quantum_reg_node_struct quantum_reg_node; typedef COMPLEX_FLOAT quantum_reg_node_amplitude;
```

```
typedef MAX_UNSIGNED quantum_reg_node_state;
```

```
/* The quantum register */
```

```
/* The quantum register */
```

```
struct quantum_reg_struct
```

```
{  
    int width; /* number of qubits in the qureg */
```

```
    int size; /* number of non-zero vectors */
```

```
    int hashw; /* width of the hash array */
```

```
    quantum_reg_node *node;
```

```
    int *hash;
```

```
};
```

inlining



```
struct quantum_reg_struct
```

```
{  
    int width; /* number of qubits in the qureg */
```

```
    int size; /* number of non-zero vectors */
```

```
    int hashw; /* width of the hash array */
```

```
    quantum_reg_node_amplitude *node_amplitude;
```

```
    quantum_reg_node_state *node_state;
```

```
    int *hash;
```

```
}
```

Transformations - struct inlining, re/de/allocation and accesses

```
diff ./qureg.c ../src/qureg.c
71,72c71,73
< reg.node = calloc(size, sizeof(quantum_reg_node));
< if(!reg.node)
---
> reg.node_amplitude = calloc(size, sizeof(quantum_reg_node_amplitude));
> reg.node_state = calloc(size, sizeof(quantum_reg_node_state));
> if(!reg.node_state || !reg.node_amplitude)
77c78,79
< quantum_memman(size * sizeof(quantum_reg_node));
---
> quantum_memman(size * sizeof(quantum_reg_node_amplitude));
> quantum_memman(size * sizeof(quantum_reg_node_state));
417c431
< j = quantum_get_state(reg1->node[i].state, *reg2);
---
> j = quantum_get_state(reg1->node_state[i], *reg2);
```

```
diff ./qureg.c ../src/qureg.c
192,194c196,201
< free(reg->node);
< quantum_memman(-reg->size * sizeof(quantum_reg_node));
< reg->node = 0;
---
> free(reg->node_state);
> free(reg->node_amplitude);
> quantum_memman(-reg->size * sizeof(quantum_reg_node_state));
> quantum_memman(-reg->size * sizeof(quantum_reg_node_amplitude));
> reg->node_state = 0;
> reg->node_amplitude = 0;
```

- quantum_memman - does not escape, in matrix.c
- calloc, malloc, free - escape, treated specially
- condition - duplicated, treated specially
- sizeof - replaced too early

Integration into LTO/WOPR

- based on gcc version 4.9.0
- 3 stages:
 - data collection (LGEN):
 - collect all symbols related to all structs, check for escapes (user defined malloc, cast, pointer arithmetic, bit fields, etc.)
 - propagation (WPA)
 - check for symbols with resolution that is not LDPR_PREVAILING_DEF_IRONLY
 - analyse: hot/cold, closeness of field accesses, with and w/o profile
 - generate transformation decisions:
 - peeling with and w/o removing of struct frame, splitting, reordering, struct-inlining
 - generate new types
 - virtually transform functions with struct params
 - variable_transfrom and function_transfrom (LTRANS):
 - varibale_transform: generate new variables (globals+static) from new types
 - function_transfrom:
 - generate new locals and temporaries
 - collect/transform all struct and field accesses
 - collect/transform all allocation sites
 - finalize transform of function calls/parameters

Integration into LTO/WOPR - problems

1. `sizeof()` - substituted by number too early, does not appear at LTO level
2. `variable_transform()` - was defined in `ipa_opt_pass_d` class (`tree-passes.h`), but was not implemented in pass manager (`passes.c`)
3. artificial property was given to new variables - `DECL_PRESERVE_P`, otherwise removed
4. `compile()` (`cgraphunit.c`) was extended with `execute_all_ipa_var_transforms()` function,
 - require review from Jan

Escape analysis - no escape example

```
struct ccount {  
    int account_number;  
    char *first_name;  
    char *last_name;  
    float balance;  
};
```

peeling

```
int foo(struct account *);
```

```
struct account acc;  
struct account *p = &acc;
```

```
foo(p);
```

<- no escape, if foo() is
PREVAILING_DEF_IRONLY

```
struct ccount_1 {int account_number;}  
struct ccount_2 {char *first_name;}  
struct ccount_3 {char *last_name;}  
struct ccount_4 {float balance;}
```

```
int foo(  
    struct account_1 *,  
    struct account_2 *,  
    struct account_3 *,  
    struct account_4 *);
```

```
struct account_1 acc_1;  
struct account_2 acc_2;  
struct account_3 acc_3;  
struct account_4 acc_4;
```

```
char * p_1 = &acc_1;  
char * p_2 = &acc_2;  
char * p_3 = &acc_3;  
char * p_4 = &acc_4;
```

```
foo (p_1, p_2, p_3, p_4);
```

Escape analysis - escaping examples

```
struct ccount {  
    int account_number;  
    char *first_name;  
    char *last_name;  
    float balance;  
};
```

```
int foo(void *);
```

```
struct account acc;  
void *p = (void *)&acc;
```

```
foo(p);
```

<- escape, even if foo() is
PREVAILING_DEF_IRONLY

```
struct ccount {  
    int account_number;  
    char *first_name;  
    char *last_name;  
    float balance;  
};
```

```
int foo(void *);
```

```
struct account acc;  
char *p = (char *)&acc;
```

```
char ch = *(p+1);
```

<- escape, pointer
arithmetic

cast from struct, or any field of struct, to other type symbolizes escape

Current status - data collection and analysis

Data collection/analysis	peeling	peeling with frame removal	reordering for struct	reordering for object	struct inlining	splitting
data collection						
sub-structs/graph of struct relations/cycle detection			n/a			
escape due to linker info						
escape due to cast and pointer arithmetic						
detect user def malloc						
analyze closeness of accesses						
prepare transform of function params			n/a			
support profile info						

Current status - transform

Transform	peeling	peeling with frame removal	reordering for struct	reordering for object	struct inlining	splitting
generate new types/new variables						
support malloc/free			n/a			n/a
generate new general accesses		n/a	n/a			n/a
generate new individual field accesses by value						
generate new individual field accesses by pointer						
generate new array field accesses by value						
generate new array field accesses by pointer						
finalize transform of function params			n/a			n/a
duplication of “if” condition						

Discussion/Plan

- Personal plan:
 - finish inlining
 - upgrade to current mainline and
 - put it on struct-reorg-branch
 - on mainline in ~year
- All are welcome!